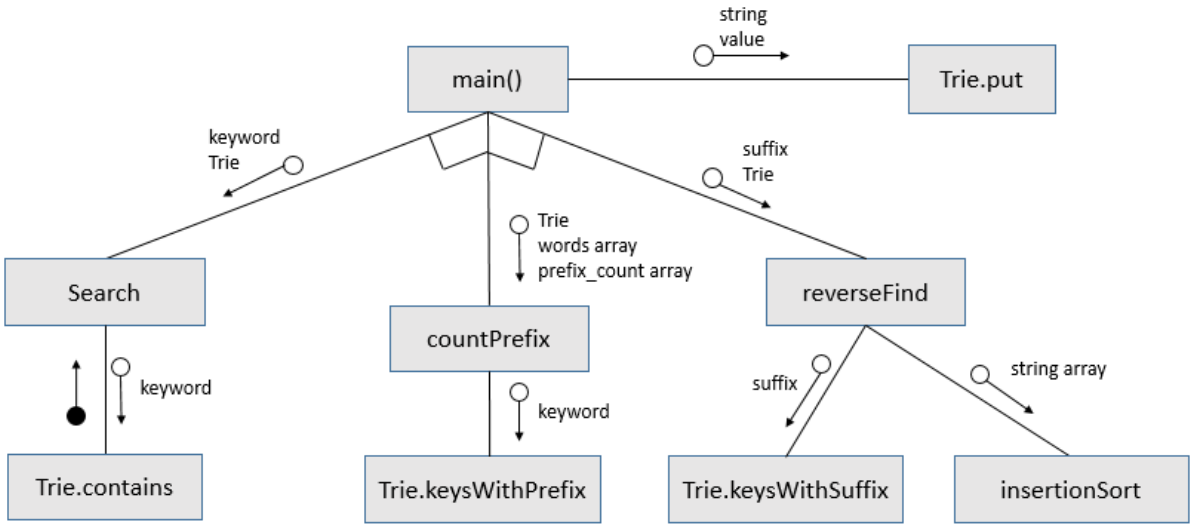
Report

Question 1

➔ Problem Statement and Code Design



➔ Implementation, Functionality

```
boolean Search (String keyword, TrieST Trie)
 {
      if (Trie contains keyword)
        {
           return true;
        }

       else
        {
           return false;
        }
}
```

```
void countPrefix (TrieST Trie, String [] strings, int [] prefix_count)
{
   for ( int i = 0; i < number of strings ;  i++)
      {
         for ( each key with prefix of the word in strings[] )
              {
                 increment prefix_count[ i ] by 1
              }

          decrement prefix_count[ i ] by 1
      }
}
```

```
boolean reverseFind (String suffix, TrieST Trie)
 {
      int count = 0;

       for (each key with suffix)
          {
              count++
          }

        initialize String array lexicographic_order with size count
        count = 0

        for (each key with suffix)
          {
              lexicographic_order [count] = key
              count++
          }

         if (count = 0)  print no result
         else sort lexicographic_order and print its elements
}
```

Search: this method searches the Trie for a given string and taking the input string and the Trie tree as parameters by calling the ***"contains"*** method in the TrieST data structure and returns true if the string was found or false, otherwise

countPrefix: this method checks if a string is a prefix of any other string in the Trie. It takes the Strings from in the same order they were read in and calls the ***"keysWithPrefix"*** method from theTrieST class then counts the number of occurrences of each string and assigns the number to the ***"prefix_count"*** array, accordingly

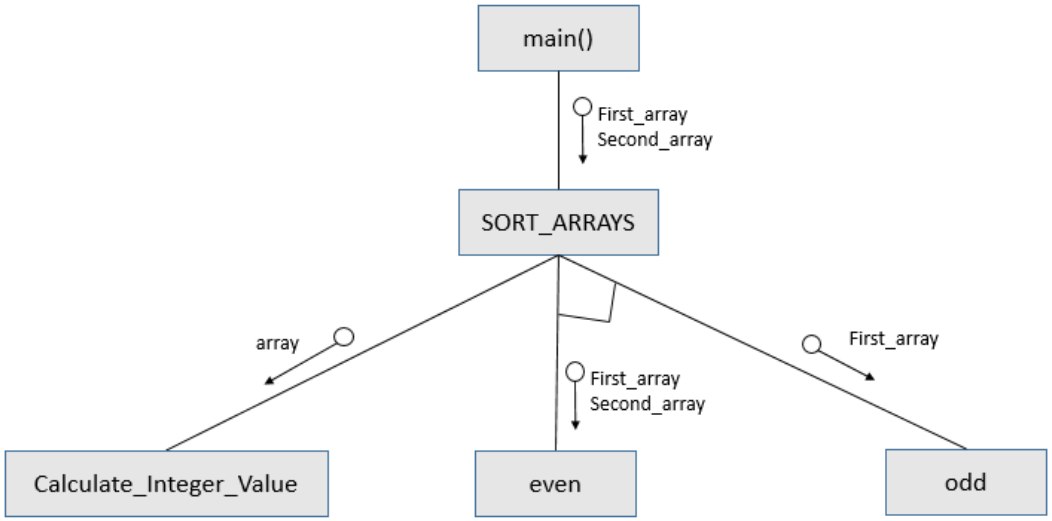
reverseFind: this method prints all the strings that end with a given suffix in the Trie, lexicographically. It counts the number of words with the given suffix and creates an array according to the count number. Then, it adds the suffixes to the initiated array by calling the ***"keysWithSuffix"*** method from theTrieST class. Next, it sorts the array using insertion sort and prints out the strings if there exists any strings.

➔ Testing

For testing this class I tried different string inputs of different sizes (different number of words and a variety of different strings) in order to validate that the methods work for different input sizes. I also tried different inputs for the 3 cases, for example I tried to change the suffix inputs and checked if the search and prefix count outputs are consistent with the input strings.

Question 2

➔ Problem Statement and Code Design



➔ Implementation, Functionality

SORT_ARRAYS: this method sorts the array based on the proposed rules. The method takes the input string arrays (First array and Second array) as parameters, and for each parallel string in the First and Second arrays, it calculates the integer value of the strings by calling the ***"Calculate_Integer_Value"*** method and gets the minimum absolute difference. If the result is even, it calls the ***"even"*** method. Else, if the result is odd it calls the ***"odd"*** method.

Calculate_Integer_Value: this method calculates the integer value of a given string. It returns a long which holds the value of that string.
even: the method takes the first string and second string as parameters and sorts the input string (first string) based on the given character order of the second string. It converts the input string into a character array and uses a custom sorting algorithm to compare the characters based on their priorities. The characters are swapped if their priorities are out of order.
odd: it takes a string as a parameter and sorts it lexicographically using a standard sorting algorithm.

```
String [] SORT_ARRAYS (String [] First_array , String [] Second_array)
 {
     initialize an array Sorted_array to store sorted array

      for (int i = 0 ; i < length of Sorted_array ; i++)
   {
     int first = calculate integer value of First_array
     int second = calculate integer value of Second_array

      if ( absolute value of first – odd is even)
        {
          Sorted_array[i] = even (String [] First_array ,
                                  String []   Second_array)
        }

       else
       {
          Sorted_array[i] = First_array[i] sorted lexicographically
       }
   }
   return Sorted_array
}
```

```
String even (String first_string , String second_string)
 {
   Convert the first string to a character array for sorting

   for (int i = 0 ; i < length of character array ; i++)
   {
     for (int j = 0 ; j < length of character array ; j++)

          char char1 = character array [i];
          char char2 = character array [j];
          int index1 = index of char1 in second_string
          int index2 = index of char2in second_string

     if (char1 or char2 is not found in the order string) {
          assign it a higher index
       }

      if (index1 > index2) {
          Swap the characters
       }
  return sorted_string
}
```

➔ Testing

For testing this class I tried the code with different array inputs. For example, I tried entering strings of different sizes and checked that the algorithm works for various input sizes. I also tried to change the content of the strings to validate the integer value calculation, the comparisons and the odd and even sorting methods function correctly.

➔ Final Assessments (For All Questions)

The trouble points in completing this assignment were mostly trying to find algorithmically efficient solutions. For example, the most challenging part was trying to figure out how to implement a customized sorting method that sorts a string based on the criteria and rules of other strings in Question 2. However, what I liked about this assignment is that it provided me with hands-on-practice about the topics of Tries and String sorts. I learned from it new algorithms for dealing with strings like sorting and searching. I also learned the algorithmic thinking behind those methods which are a very important topic in Data Structures and Algorithms.